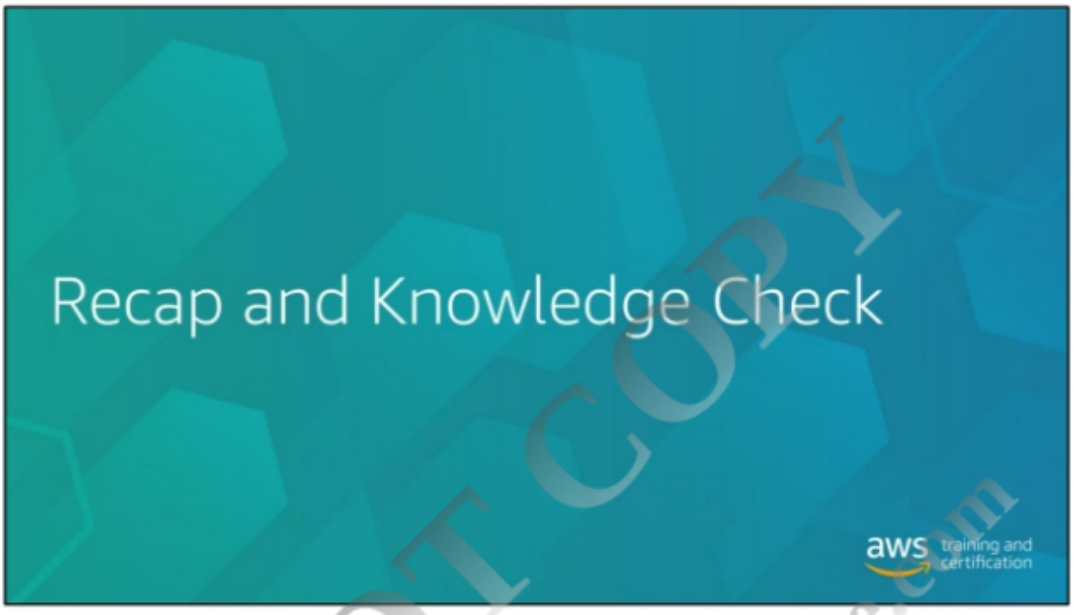
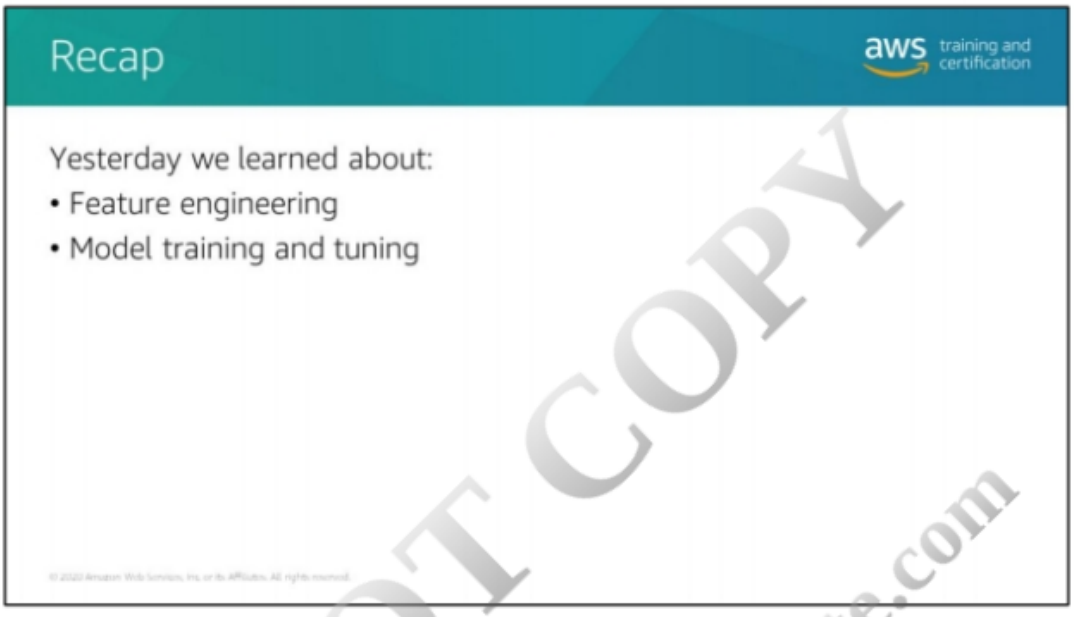
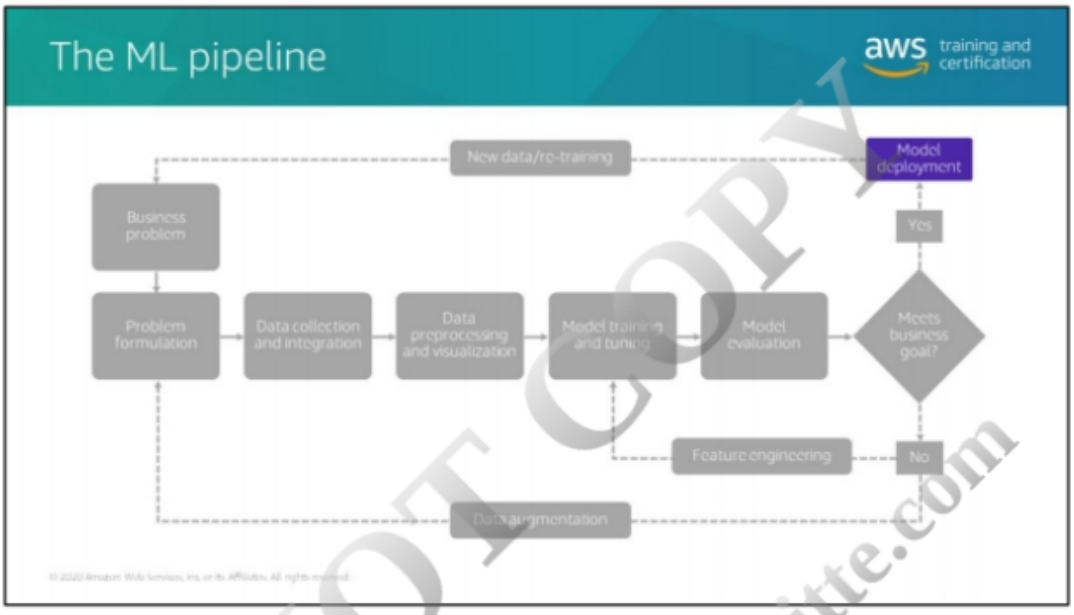


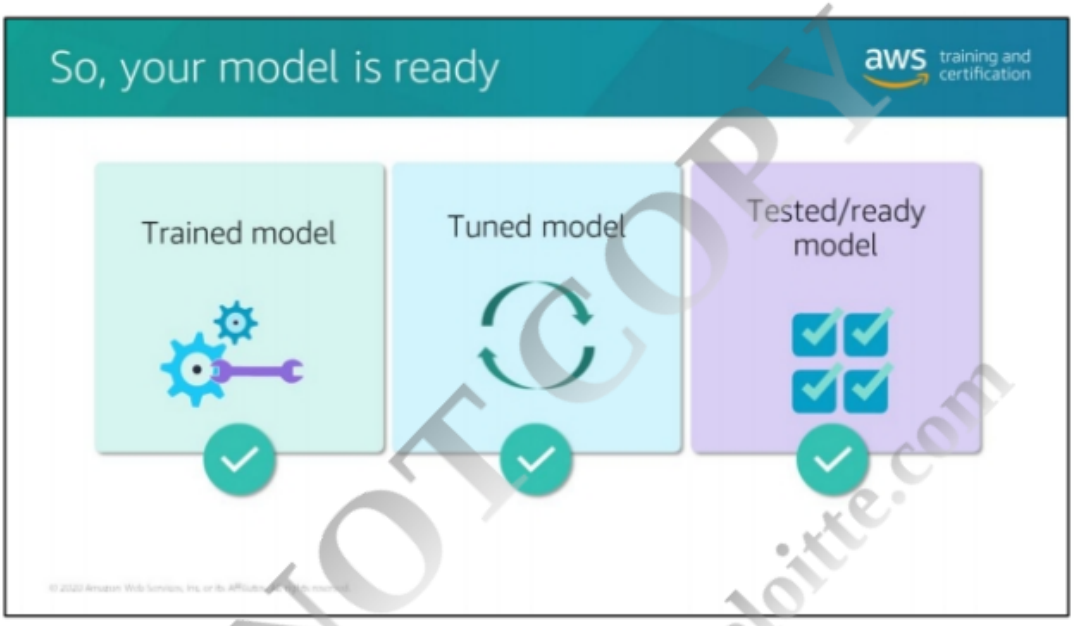


https://amazonmr.au1.qualtrics.com/jfe/form/SV_7W24cR6RoXBrZqd

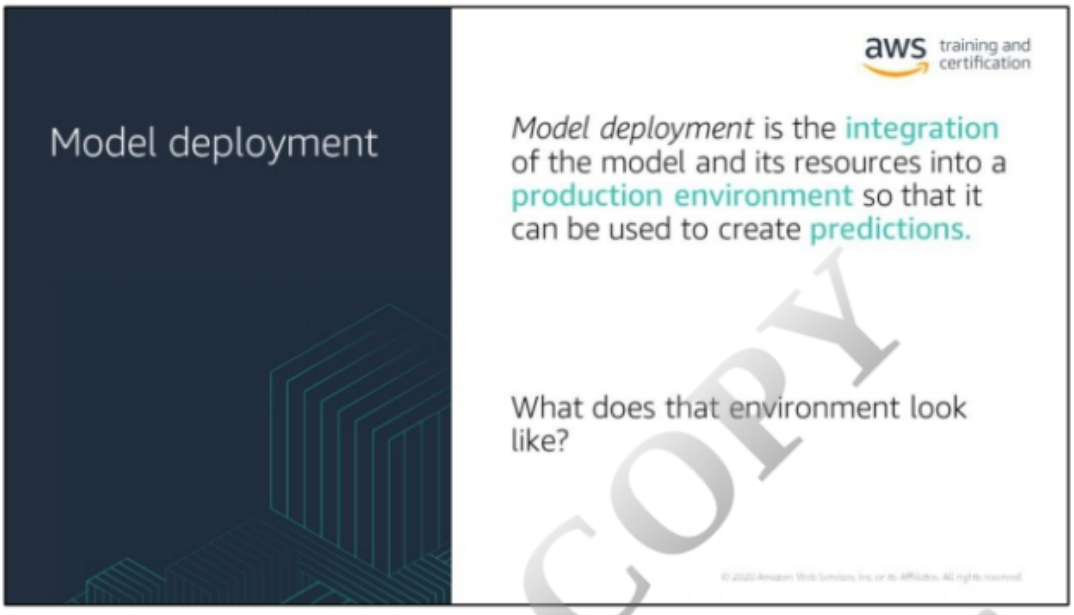


Now that you've completed engineering your features and tuning your model, the model is ready for deployment!

In this module about model deployment we will talk about inference types, monitoring models, and deployment approaches.

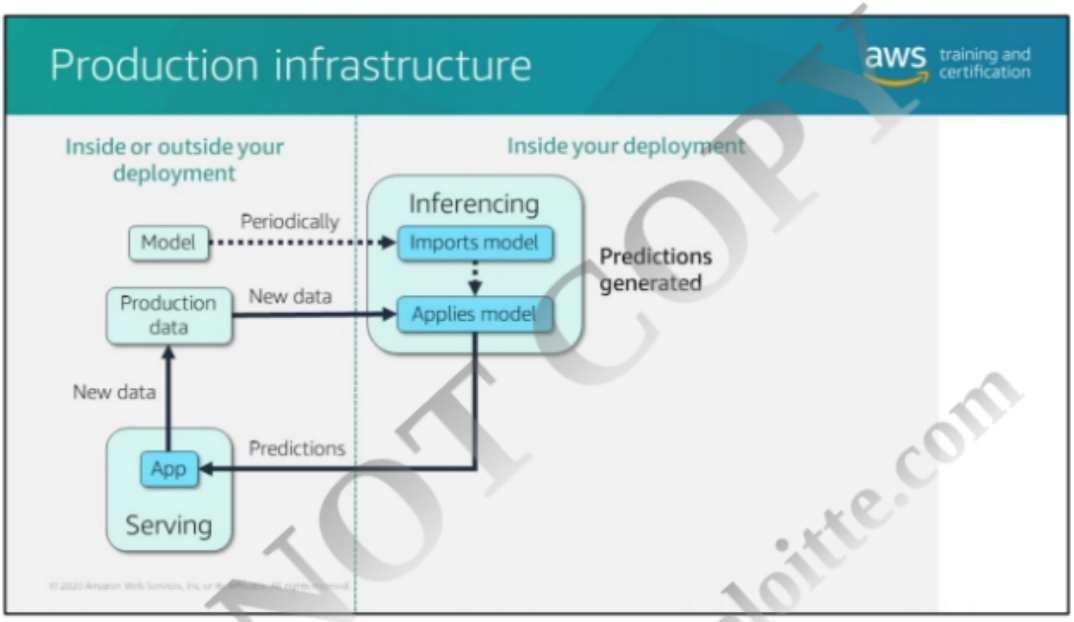


Ok now you have a model that is trained, tuned and tested. You are ready to use the model to make inferences or predictions. How do you deploy your model?

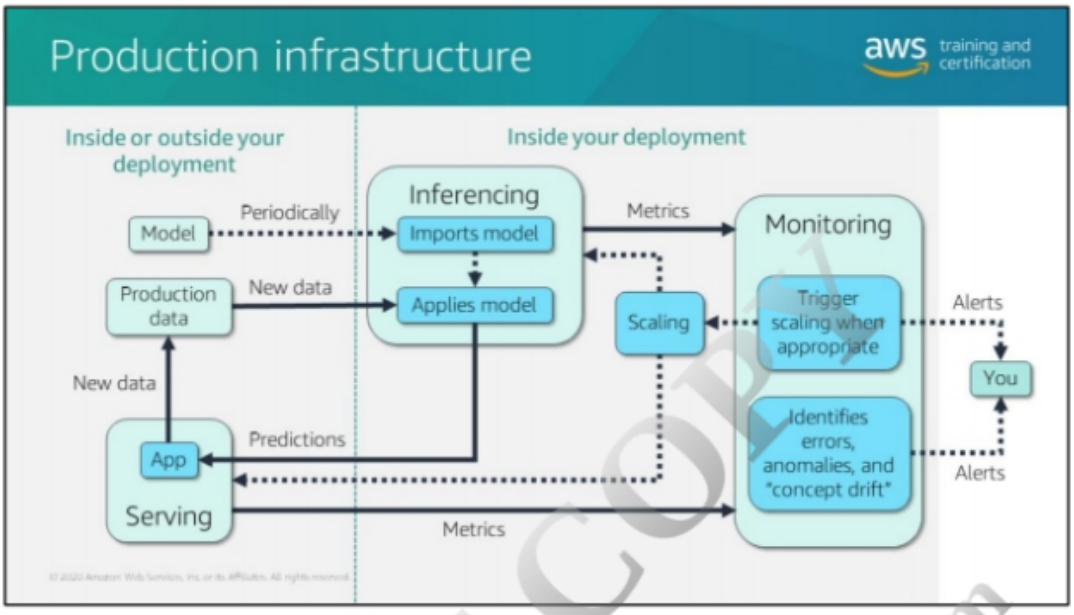


For the purposes of this module, we're defining model deployment of an ML-model as the process of integrating the model into a production environment.

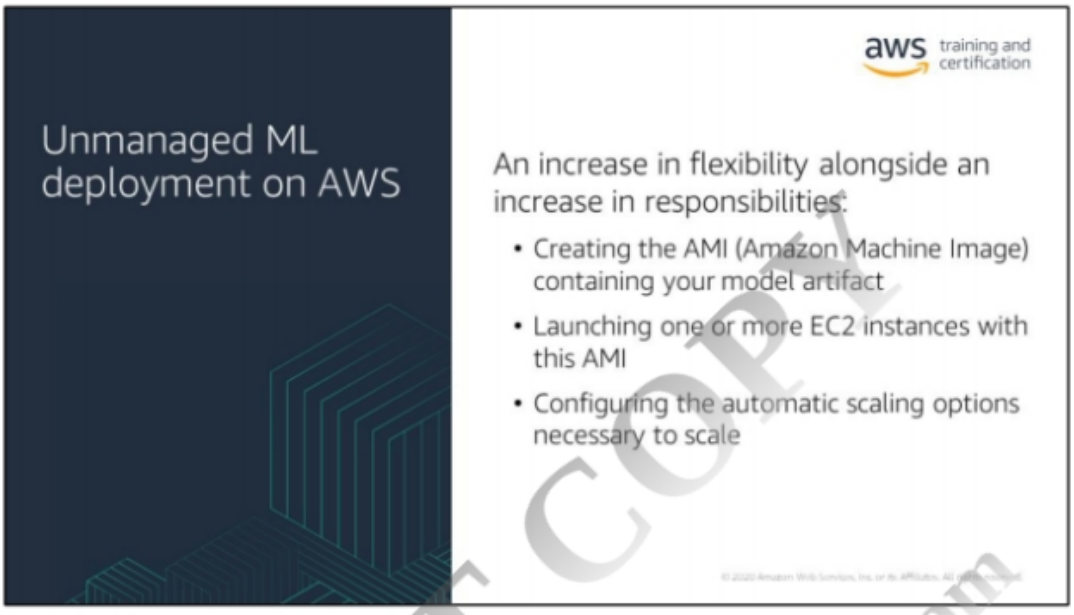
Your deployment infrastructure depends on your use case. In cases where low latency is critical, you'll want to host your production model, data, and even serving layer with your application on your deployment infrastructure even though it may cost you more to do so. But you could also store these things elsewhere, for instance your data and model could be stored in Amazon S3 while your serving layer is hosted on Amazon EC2 or Lambda. In either case, your application will pass relevant new data to where you store your production data.



This triggers your inferencing layer. Your inferencing layer will update its copy of the model periodically, such as on a schedule or if the model has been retrained. When it receives new data, the inferencing layer will apply it to the new data and generate predictions.



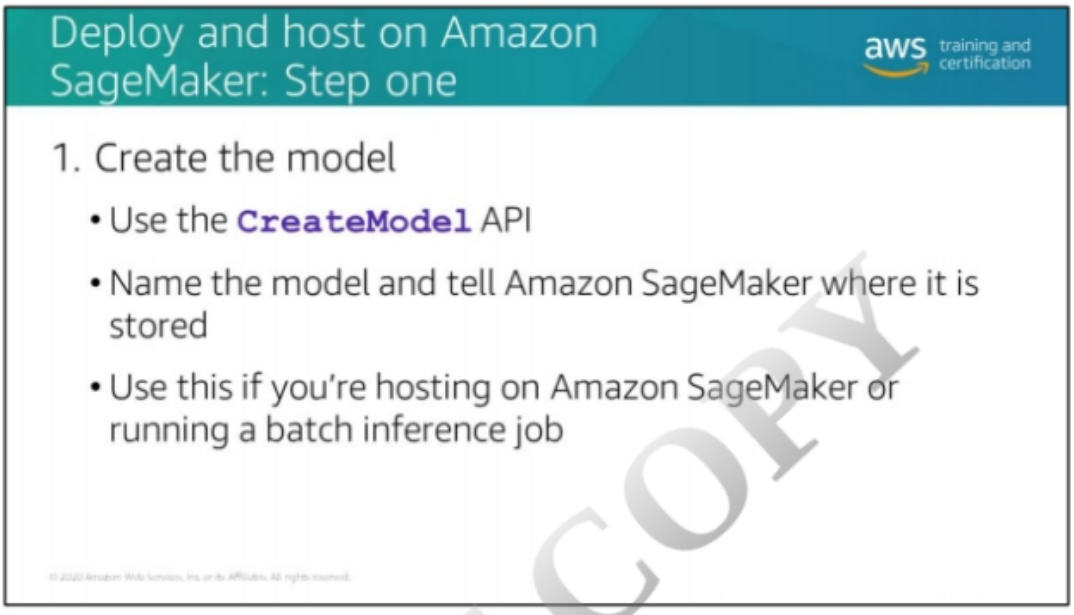
Finally, as with any infrastructure you'll need to keep an eye on it to make sure everything's working correctly and to respond when necessary. In this case, that means collecting metrics from the inferencing and serving layers, triggering scaling when appropriate, and alerting you of any scaling, errors, anomalies, or what we call "concept drift", which is something we'll get into later in the module. It's basically how models tend to slowly degrade in their accuracy over time and require retraining. We'll talk individually about each of these layers: inferencing, monitoring, and serving, but first let's get into how deploying your infrastructure works.

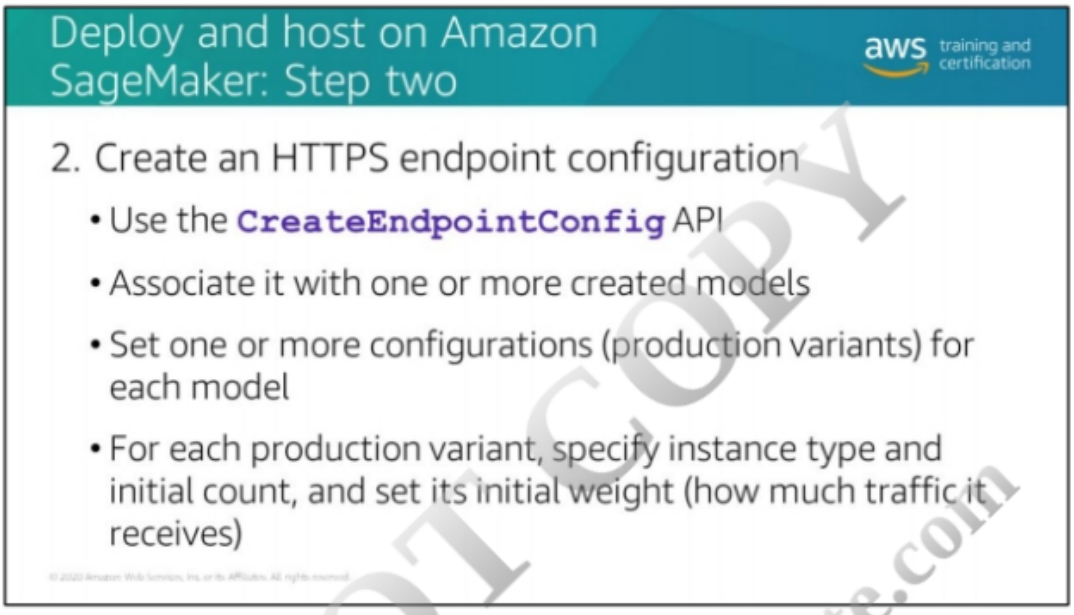


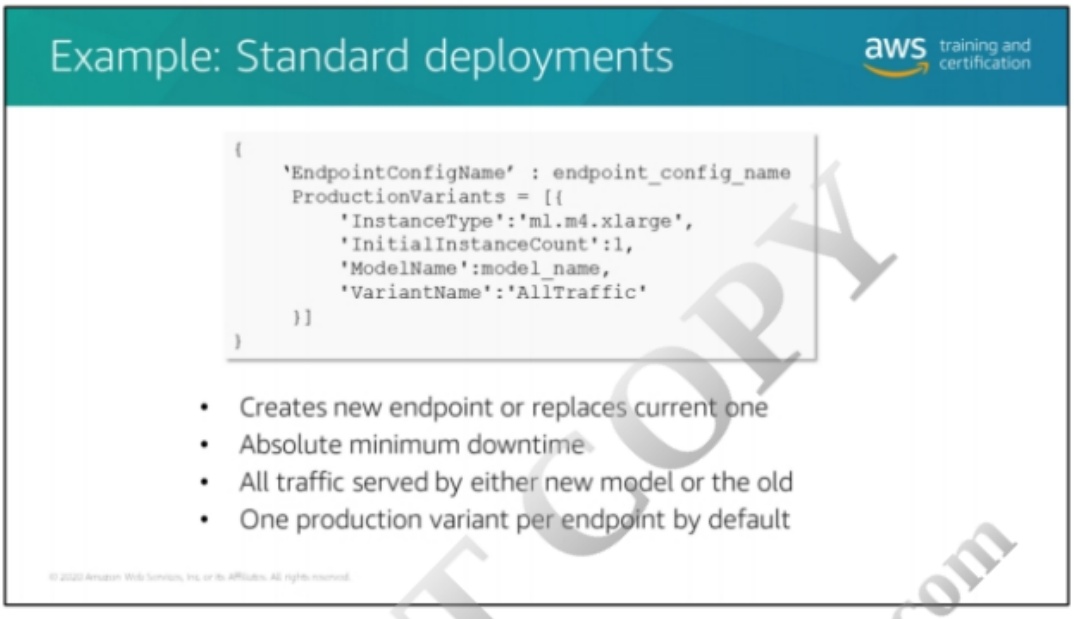
Do you have to use Amazon SageMaker to host your models on AWS? No. It is certainly possible to host ML models directly on Amazon EC2 instances or containers (like ECS or EKS) and make them available to your consumers. However, if you choose one of those options, you are responsible for creating the AMI (Amazon Machine Image) containing your model artifact, launching one or more EC2 instances with this AMI and configuring the auto scaling options necessary to scale according to the inference traffic patterns.

As you can see, this involves invoking and coordinating multiple API calls. Amazon SageMaker, on the other hand offers a "Managed Service" experience where a single API call takes care of deploying your model on a cluster of one or more instances and auto scaling the cluster to appropriately meet the inference demand from your consumers.

On AWS, Amazon SageMaker offers a broad variety of options for deployment and inference, and is the recommended service for deploying (also called hosting) your production ML models. Amazon SageMaker, as you already know is the managed platform for end to end machine learning. It provides model hosting services for model deployment, and provides an HTTPS endpoint where the ML model is available to provide inferences.



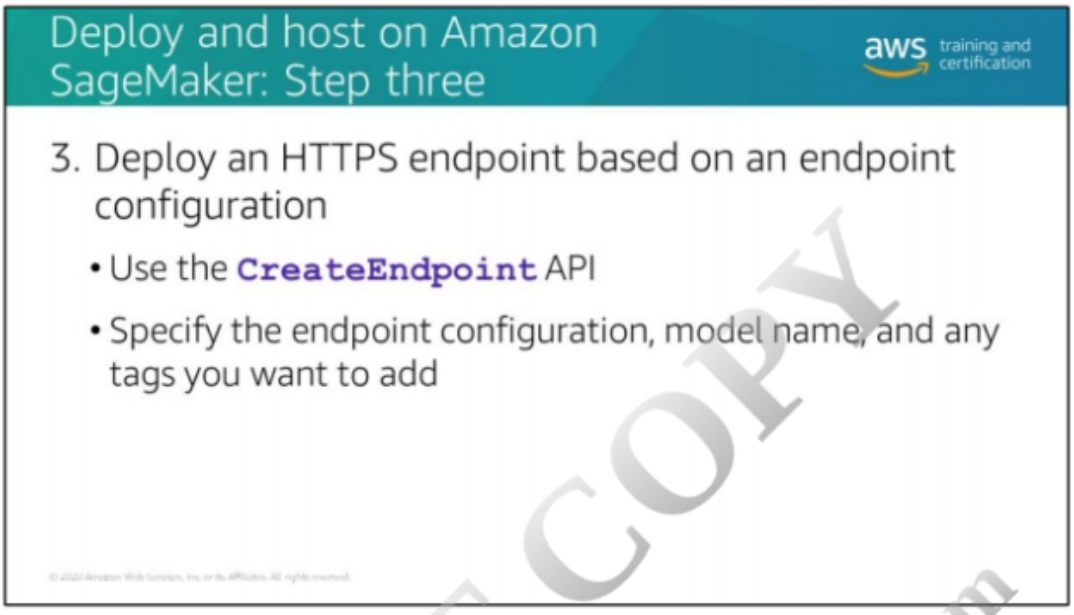


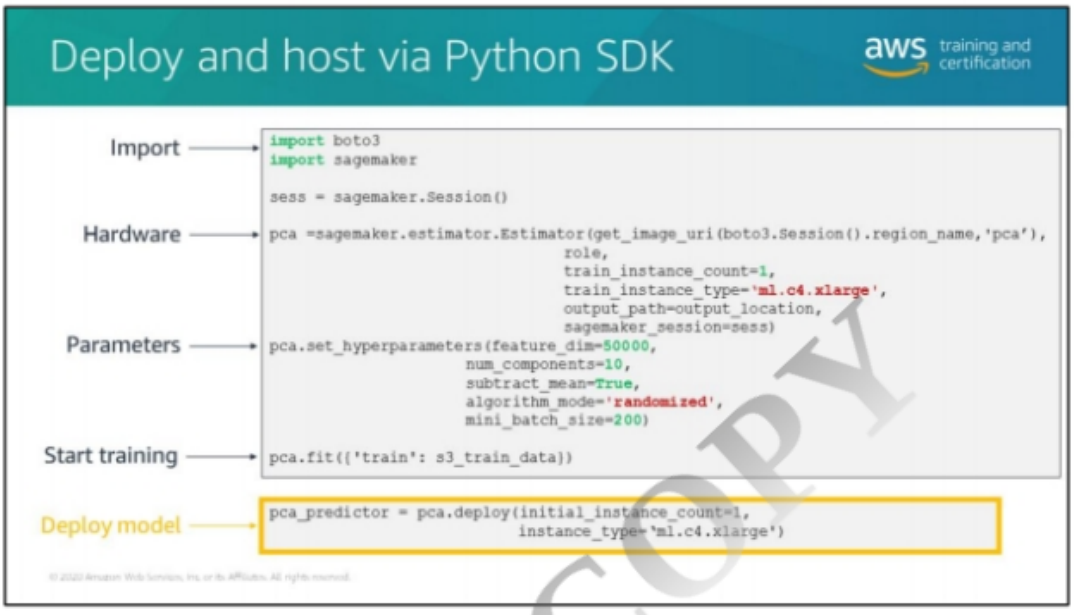


Let's look at some deployment strategies using production variants.

We have already seen an example of this. In a standard model deployment, the Amazon SageMaker endpoint is configured with a single production variant. The production variant configuration specifies the type and count of the instance to host the model. The following is a sample production variant configuration for a standard deployment. Here there is only a single model behind the end point and all inference traffic is processed by the single model.

In this approach, when you have a newer model as a result of model retraining efforts, you will update the endpoint with a new endpoint configuration pointing the newer model. Amazon SageMaker creates the required infrastructure for the new production variant and updates the weights without any downtime. All inference traffic is now served by the new model. Essentially, the inference traffic is served by either the old version of the model or the new version.

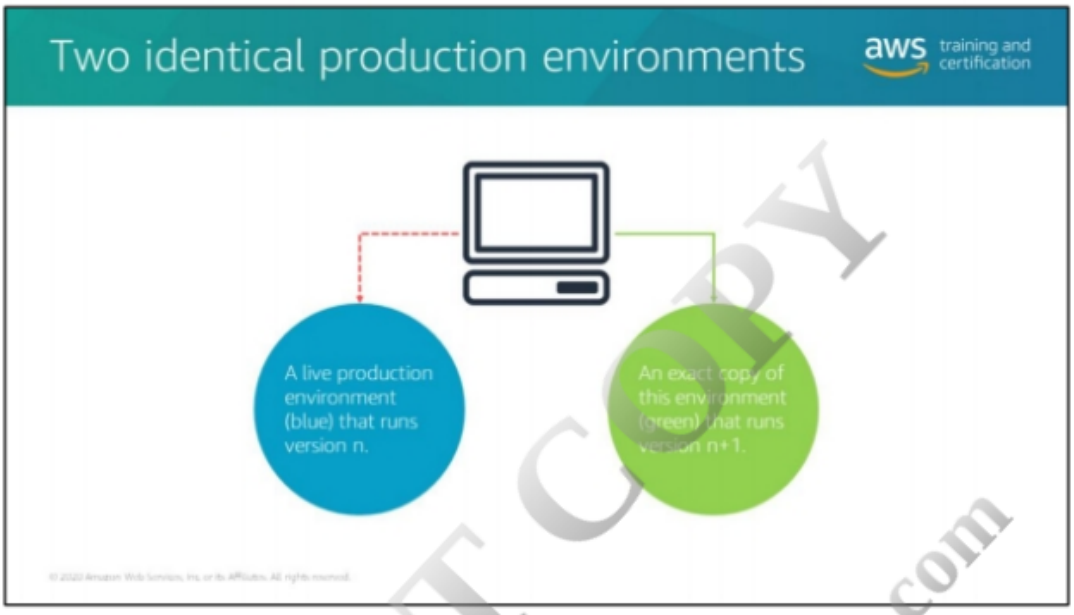




Here's the earlier example code using the PCA algorithm, this time we've added code to deploy the model.

Another important criteria is to deploy newer models to production in a way that reduces risk and minimizes downtime to the model consumers. Replacing an existing model with a newer/better model should ideally not cause any service interruption to the model consumers. Towards this goal, you can use the "production variants" capability of Amazon SageMaker.

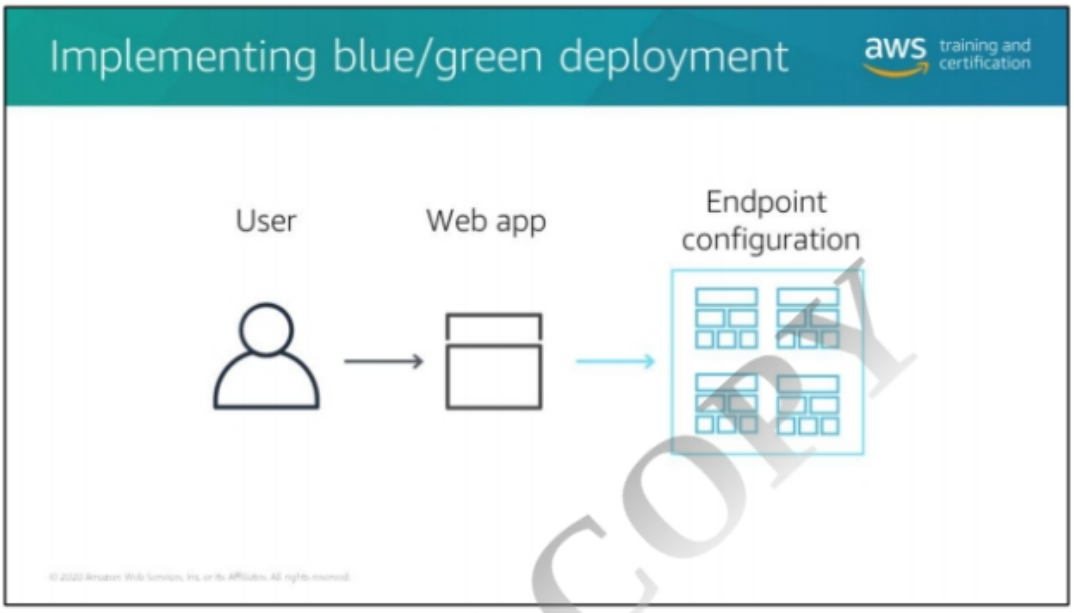
The typical A/B, Blue/Green, Canary deployment strategies that are used in the software development lifecycle production release process are also applicable in deploying the machine learning models.



The blue/green deployment technique provides two identical production environments. You can use this technique when you need to deploy a new version of the model to production. As shown in the figure, this technique requires two identical environments:

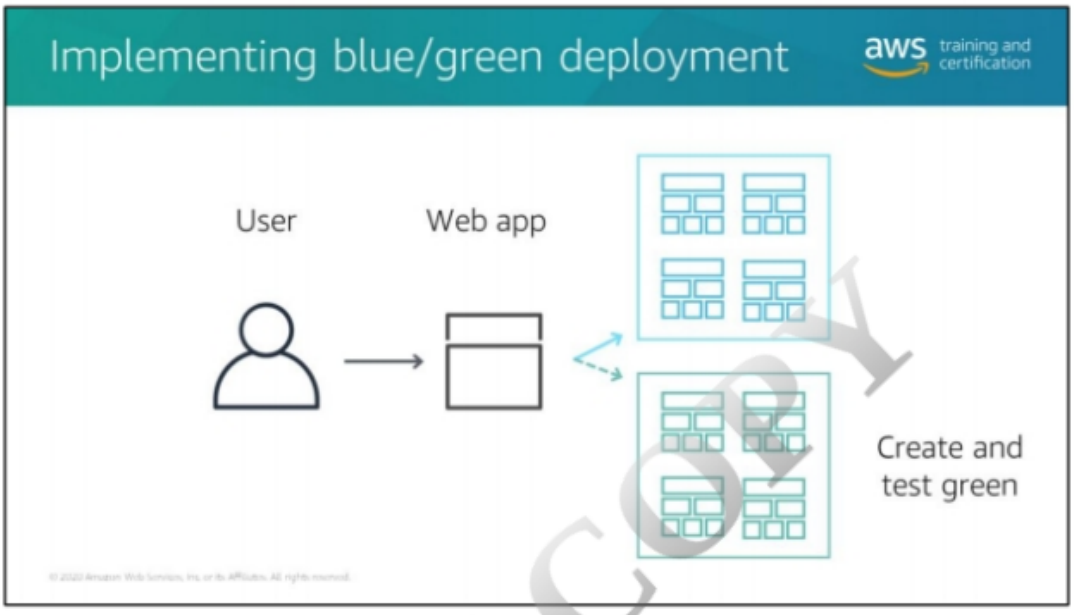
- A live production environment (blue) that runs **version n**,
- An exact copy of this environment (green) that runs **version n+1**.

While the blue environment (version n) is processing the live traffic, you test the next release (version n+1) on the green environment with **synthetic** traffic. Tests should include verifying that the new model is meeting both technical and business metrics. If all tests of version n+1 in the green environment are a success, then the live traffic is switched to the green environment. You then validate the metrics again in the green environment, this time with live traffic. If you find any issues in this testing, you switch the traffic back to blue environment. If no issues are found for a period of time, you can delete the blue environment.



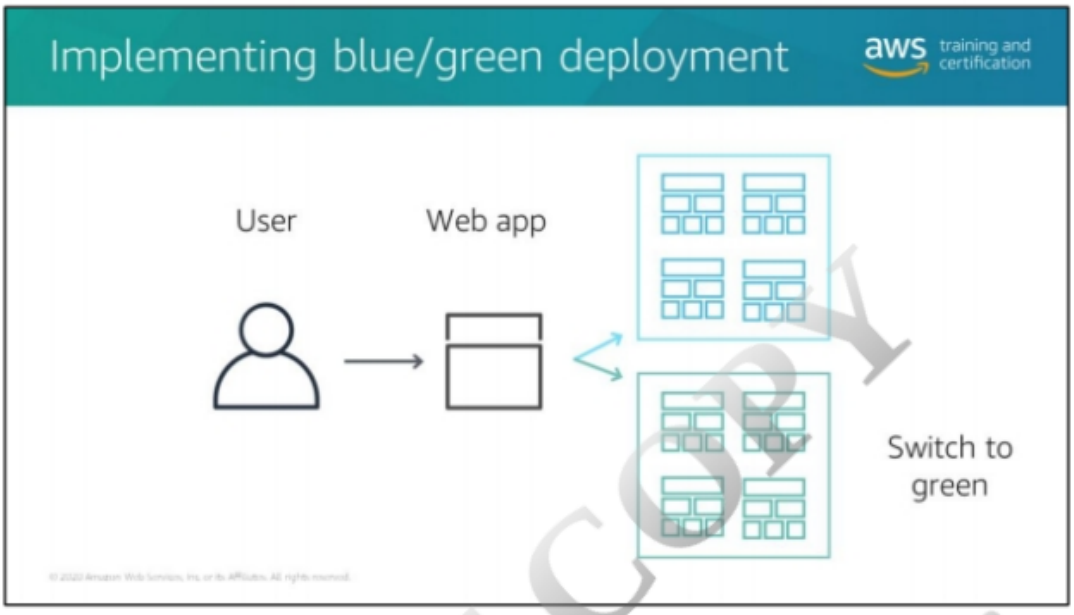
Implementing a blue/green deployment on Amazon SageMaker includes these steps:

1. Create a new endpoint configuration, using the same production variants for the existing live model and for the new model.



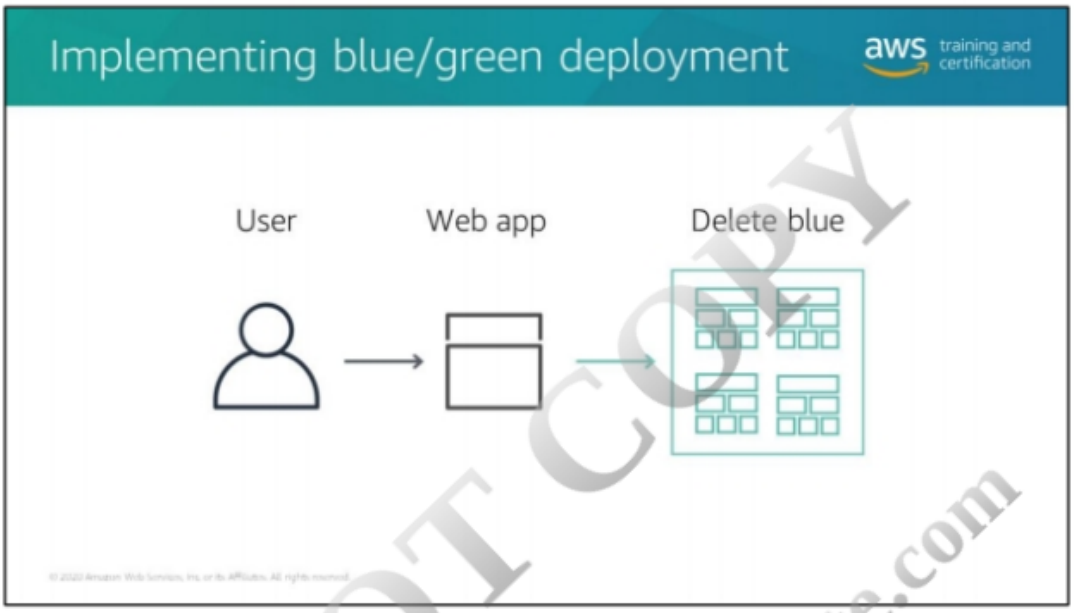
Implementing a blue/green deployment on Amazon SageMaker includes these steps:

1. Create a new endpoint configuration, using the same production variants for the existing live model and for the new model.
2. Update the existing live endpoint with the new endpoint configuration. Amazon SageMaker creates the required infrastructure for the new production variant and updates the weights without any downtime.



Implementing a blue/green deployment on Amazon SageMaker includes these steps:

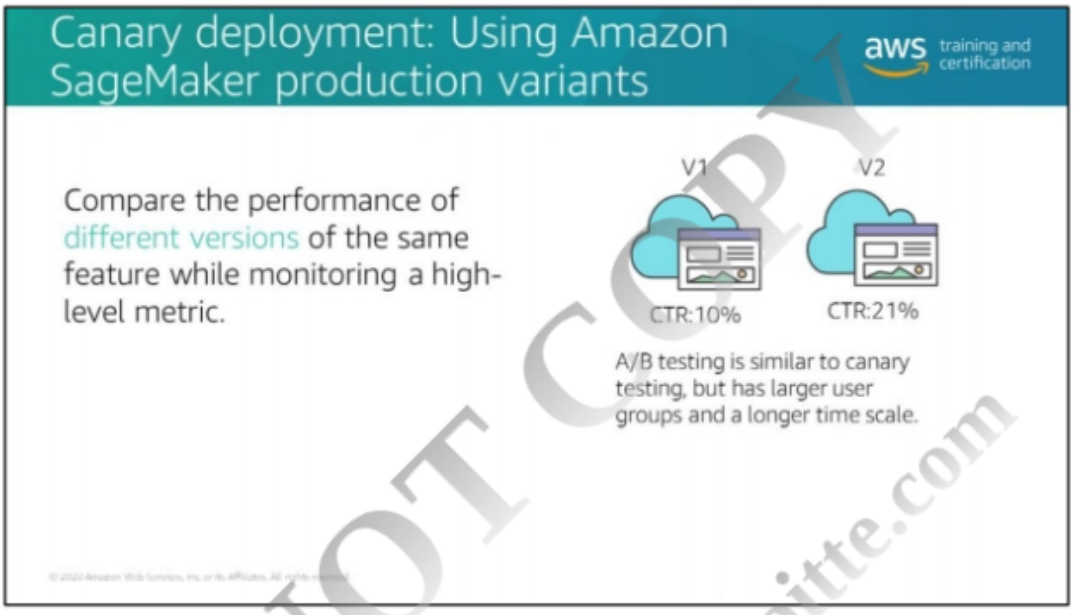
1. Create a new endpoint configuration, using the same production variants for the existing live model and for the new model.
2. Update the existing live endpoint with the new endpoint configuration. Amazon SageMaker creates the required infrastructure for the new production variant and updates the weights without any downtime.
3. Switch traffic to the new model through an API call.



Implementing a blue/green deployment on Amazon SageMaker includes these steps:

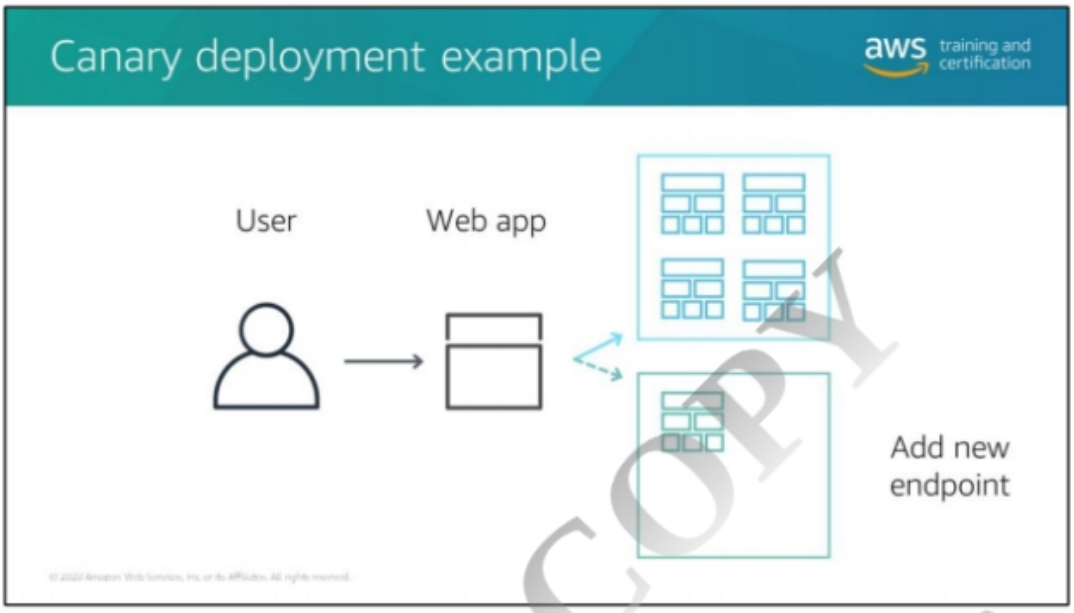
1. Create a new endpoint configuration, using the same production variants for the existing live model and for the new model.
2. Update the existing live endpoint with the new endpoint configuration. Amazon SageMaker creates the required infrastructure for the new production variant and updates the weights without any downtime.
3. Switch traffic to the new model through an API call.
4. Create a new endpoint configuration with only the new production variant and apply it to the endpoint.
   Amazon SageMaker terminates the infrastructure for the previous production variant.

In this approach, all live inference traffic is served by either the old or new model at any given point. However, before directing the live traffic to new model, synthetic traffic is used to test and validate the new model.

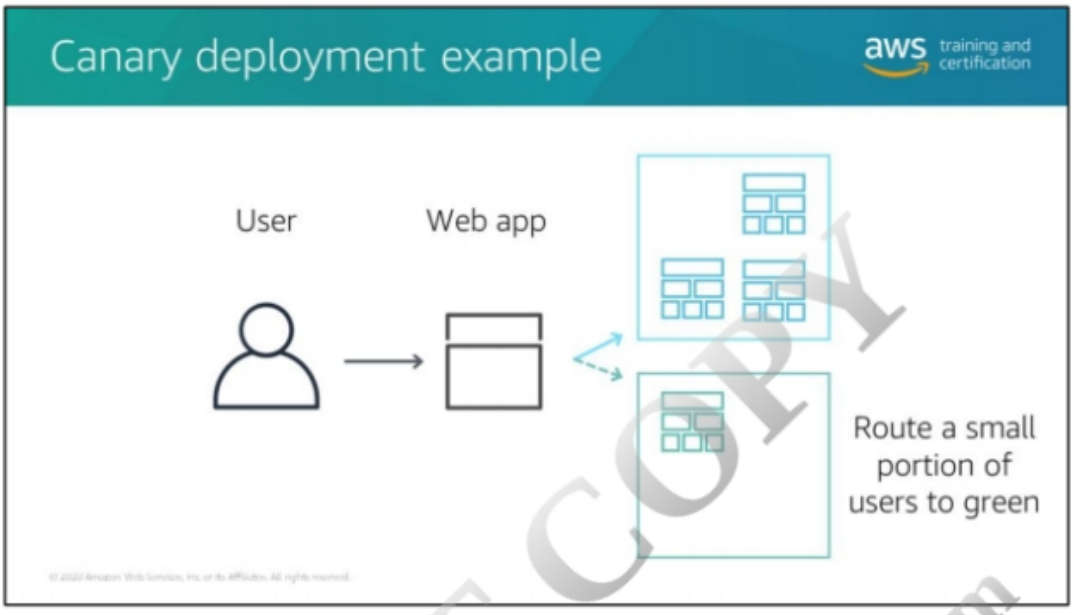


A/B testing is similar to canary testing, but has larger user groups and a longer time scale, typically days or even weeks. For this type of testing, Amazon SageMaker endpoint configuration uses two production variants: one for model A, and one for model B. For a fair comparison of two models, begin by configuring the settings for both models to balance traffic between the models equally (50/50) and make sure that both models have identical instance configurations. This initial setting is necessary so the neither version of the model is impacted by difference in traffic patterns or difference in the underlying compute capacity.

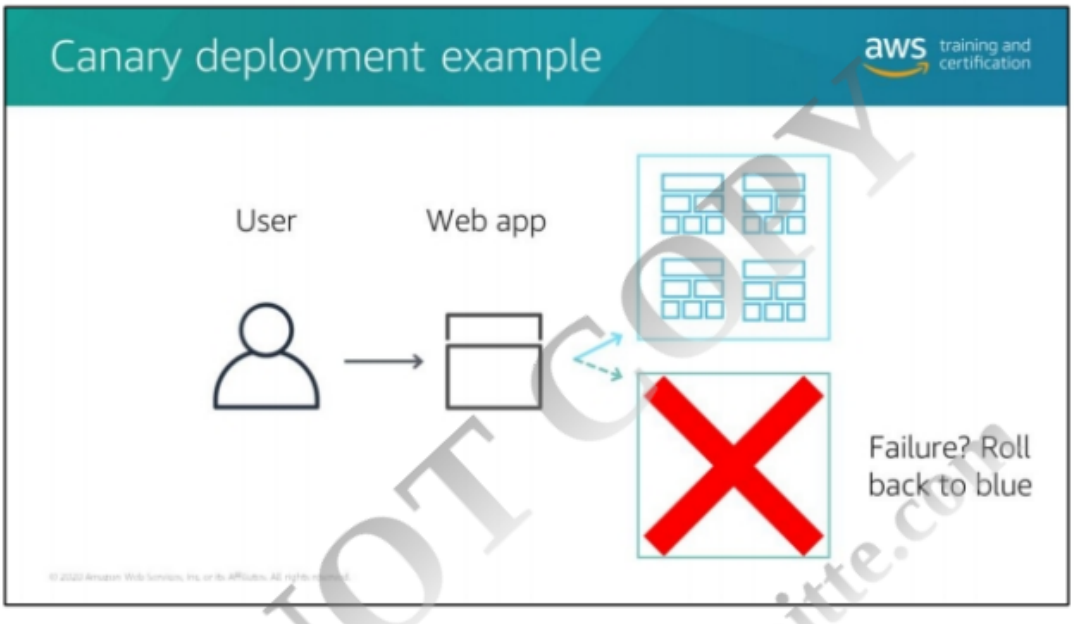
After you have monitored the performance of both models with the initial setting of equal weights, you can either gradually change the traffic weights to put the models out of balance (60/40, 80/20, etc.), or you can change the weights in a single step, continuing until a single model is processing all of the live traffic.



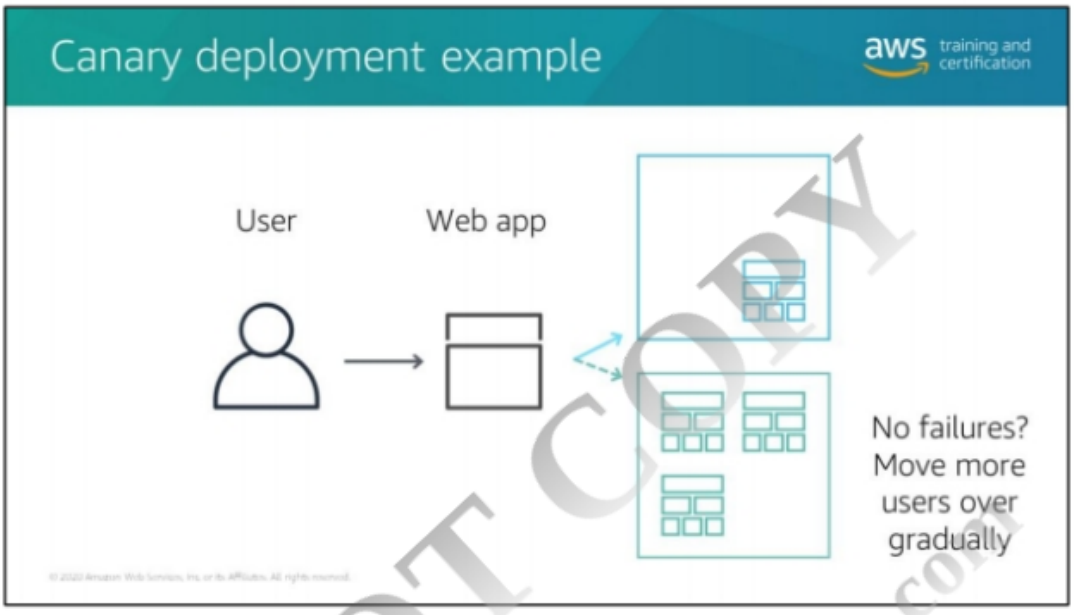
With canary testing, you can validate a new release with minimal risk



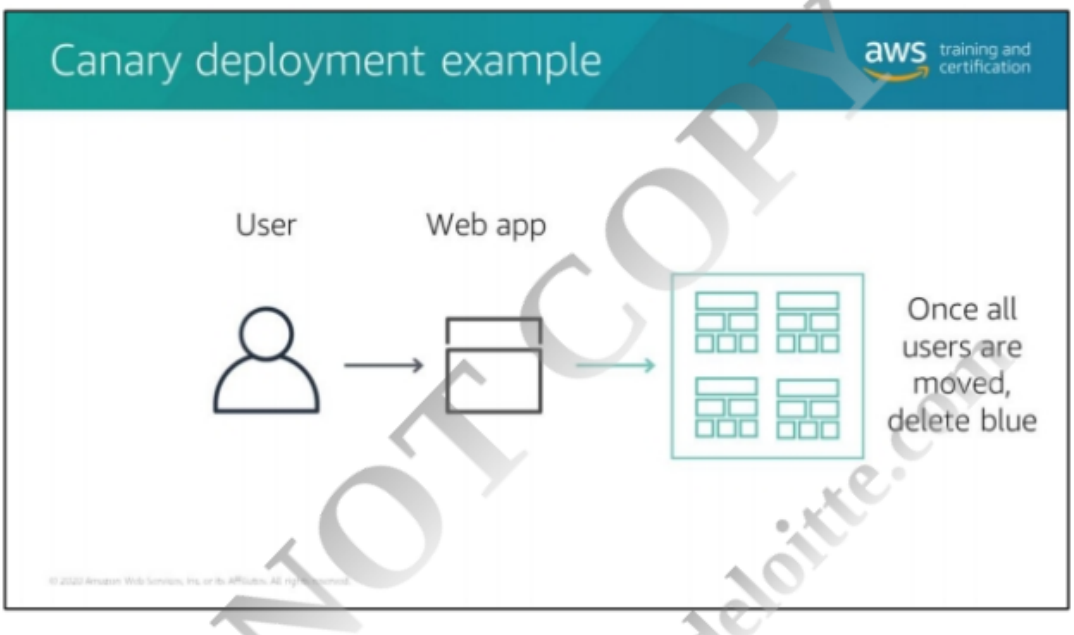
You do this by first deploying to a small group of your users. Other users continue to use the previous version.



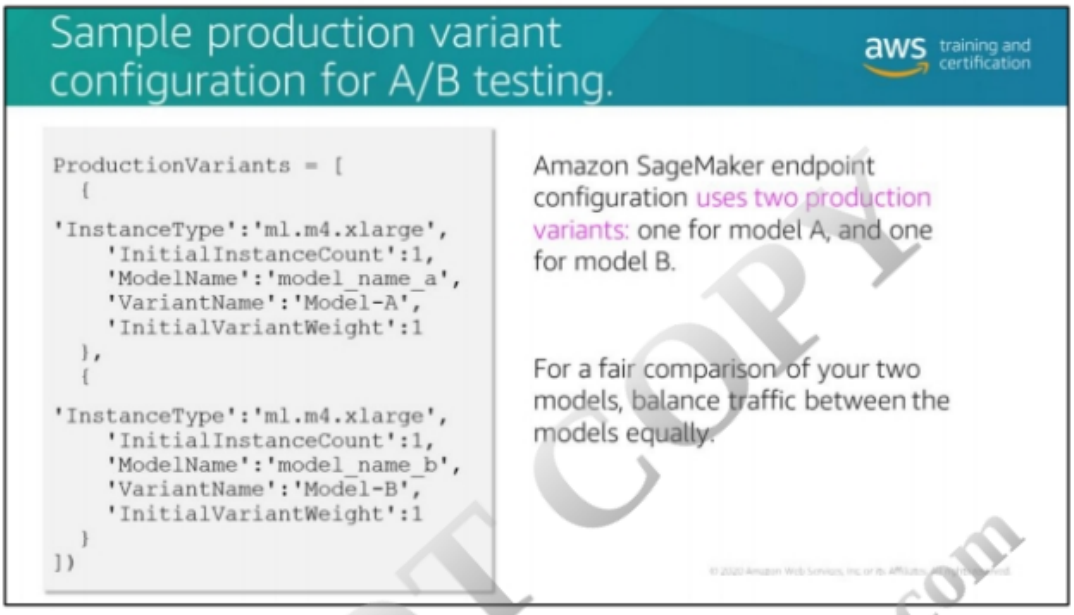
You do this by first deploying to a small group of your users. Other users continue to use the previous version.



When you're satisfied with the new release, you can gradually roll the new release out to all users.

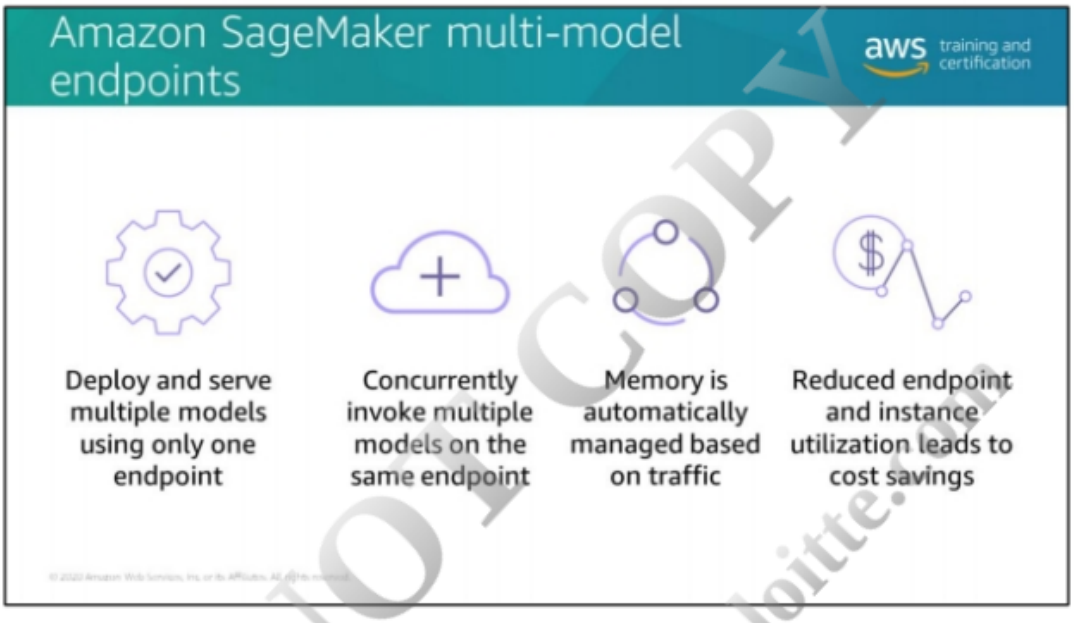


After you have confirmed that the new model performs as expected, you can gradually roll it out to all users, scaling endpoints up and down accordingly.

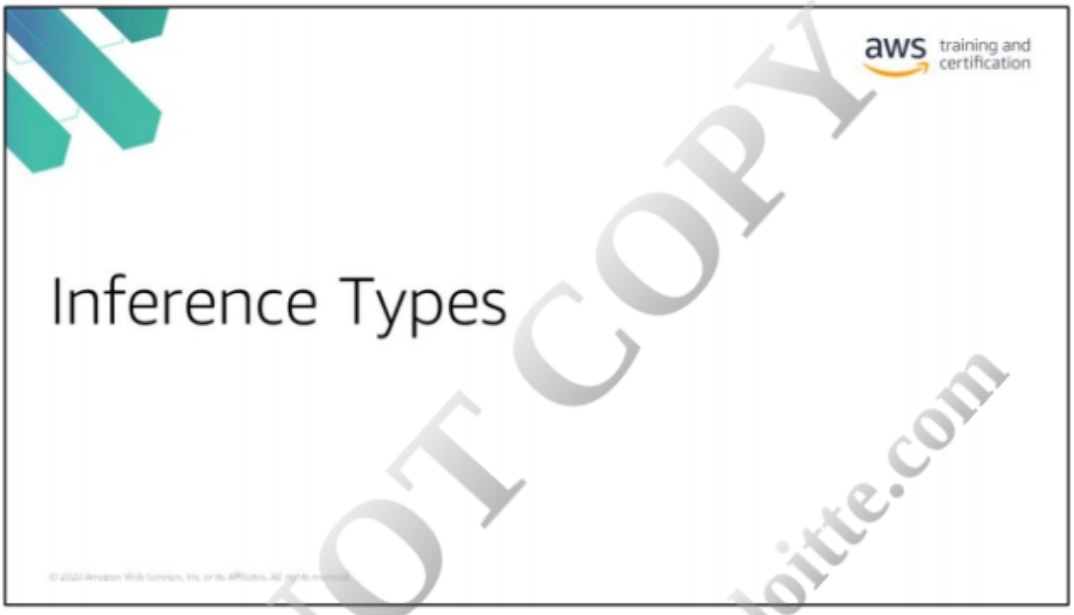


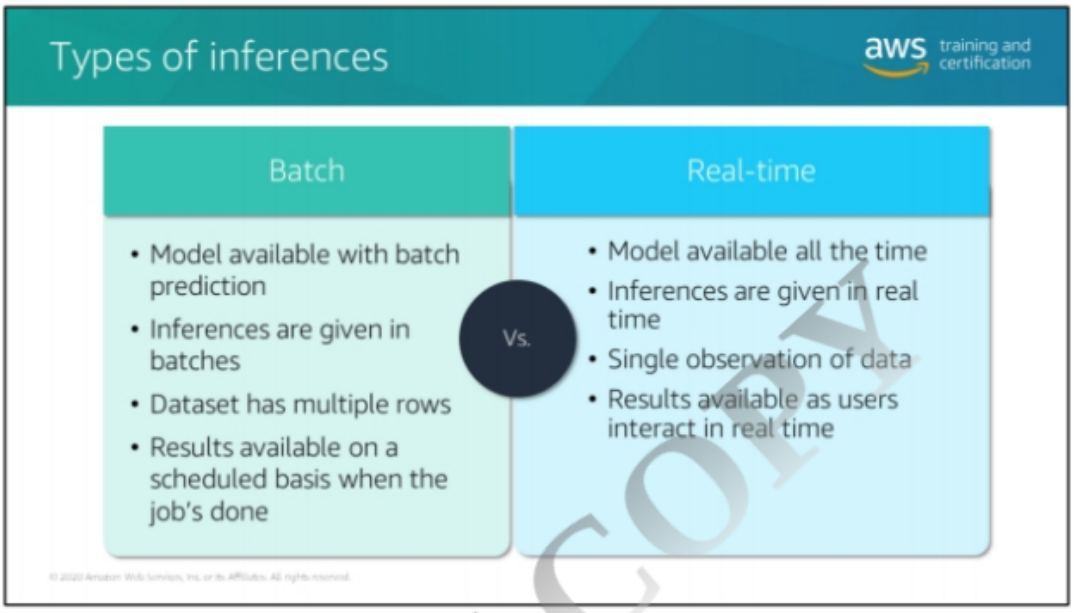
Here is a sample production variant configuration for A/B testing.

A/B testing is similar to canary testing, but has larger user groups and a longer time scale, typically days or even weeks. For this type of testing, Amazon SageMaker endpoint configuration uses two production variants: one for model A, and one for model B. For a fair comparison of two models, begin by configuring the settings for both models to balance traffic between the models equally (50/50) and make sure that both models have identical instance configurations. This initial setting is necessary so the neither version of the model is impacted by difference in traffic patterns or difference in the underlying compute capacity. After you have monitored the performance of both models with the initial setting of equal weights, you can either gradually change the traffic weights to put the models out of balance (60/40, 80/20, etc.), or you can change the weights in a single step, continuing until a single model is processing all of the live traffic.



Using Amazon SageMaker multi-model endpoints enables several important benefits. It allows you to deploy and serve multiple models using only one endpoint. You can concurrently invoke multiple models on the same endpoint. And what everyone likes, especially leadership; because the memory is automatically managed based on traffic and the reduced endpoint and instance utilization leads to cost savings





A quick reminder that you can use SageMaker to deploy a model to get predictions in 2 ways:

1. You can use Amazon SageMaker batch transform to get predictions for an entire dataset.
2. Or you can set up a persistent endpoint to get one prediction at a time using Amazon SageMaker Hosting Services.

With this in mind, consider the inference architecture needed for your machine learning model. That is, consider whether the model needs to deployed on-demand as a batch of inference requests come in, or does it need to be available 24 X 7 for real-time predictions.

For example, the "forecasting" model for the e-commerce company that was cited earlier needs to be available periodically when the inventory team needs sales projections. Such predictions are typically asynchronous and made on a dataset consisting of multiple rows of data. You submit a long running inference job and results are available when the job is completed.

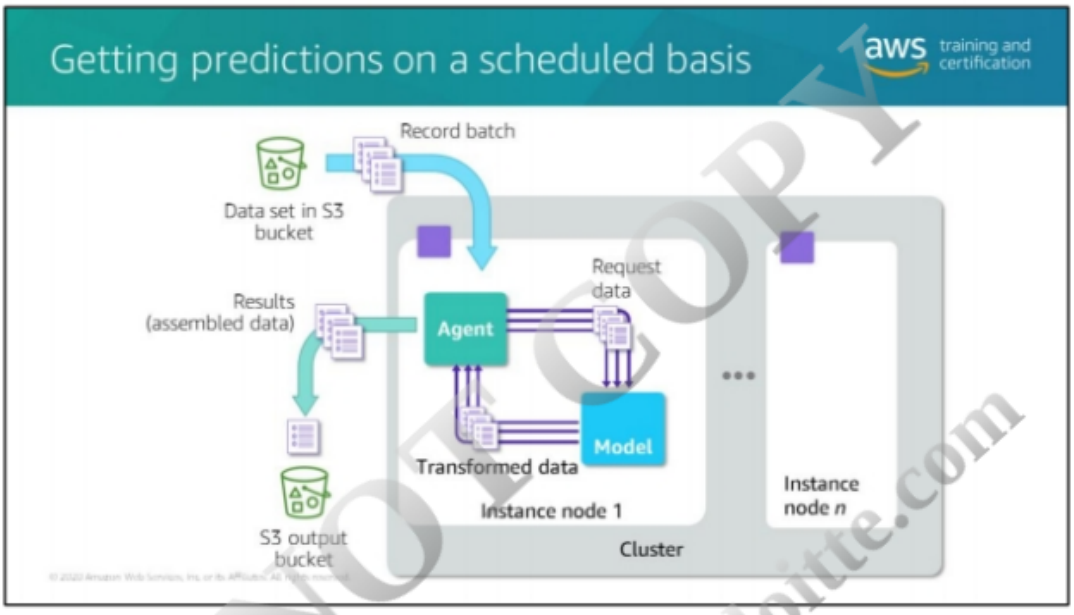
On the other, the "fraud detection" model should be available all the time and provide real-time fraud detection capability as user interact with the web application. Such predictions are synchronous, real-time and most often happen on a single observation of data.

What if you don't need low latency responses for your inferences? What if they can be performed in batches, such as for a model that predicts how many products need to be ordered and put in stock at a retail store? You most likely only need those inferences before orders need to be placed, so the inferences can be run in batches based on how products are ordered.

Now that you know about the two different ways to provide consumers with ML models for inference, when and how would you choose one over the other? Specifically, when would you use Batch Transform as opposed to endpoints?

Use batch transform when you:

- Want to get inferences for an entire dataset and index them to serve inferences in real-time
- Don't need a persistent endpoint that applications (for example, web or mobile apps) can call to get inferences
- Don't need the sub-second latency that Amazon SageMaker hosted endpoints provide
- Additionally, you can also use batch transform to preprocess your data before using it to train a new model or generate inferences.



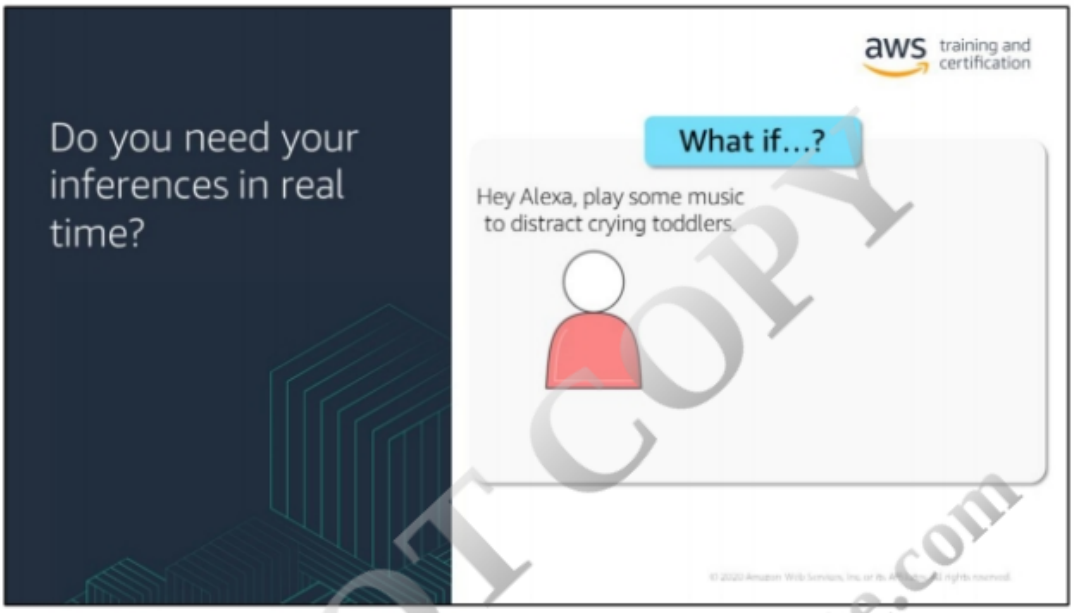
Recall our earlier discussion about endpoints and batch transform. Batch transform is for serving predictions on a scheduled basis, for instance, at a certain time of day. Endpoints are best for online inferencing when data is coming from the internet and you need to service predictions in real-time.

Let's dive a little deeper into these, beginning with batch transform. To get inferences for an entire dataset instead of for each data item, use batch transform. To perform a batch transform, create a batch transform job using either the Amazon SageMaker console or the API. Provide the following:

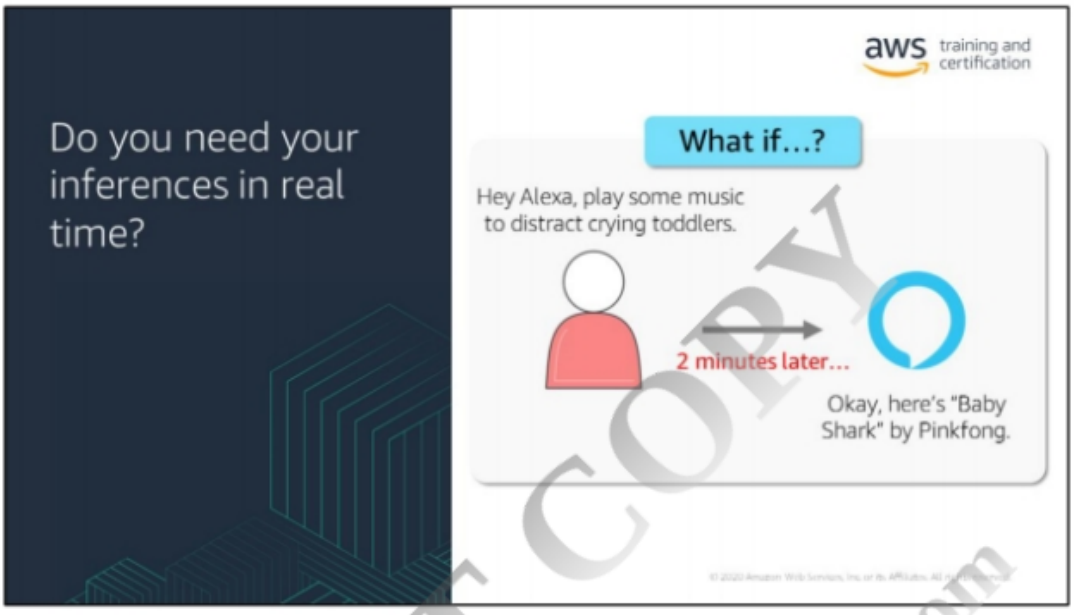
- The path to the S3 bucket where you've stored the data that you want to transform.
- The compute resources that you want Amazon SageMaker to use for the transform job.
- The path to the S3 bucket where you want to store the output of the job.
- The name of the Amazon SageMaker model that you want to use to create inferences.

Batch transform manages all of the compute resources required to get inferences. This includes launching instances and deleting them after the batch transform job has completed. Batch transform manages interactions between the data and the model with an object within the instance node called an agent.

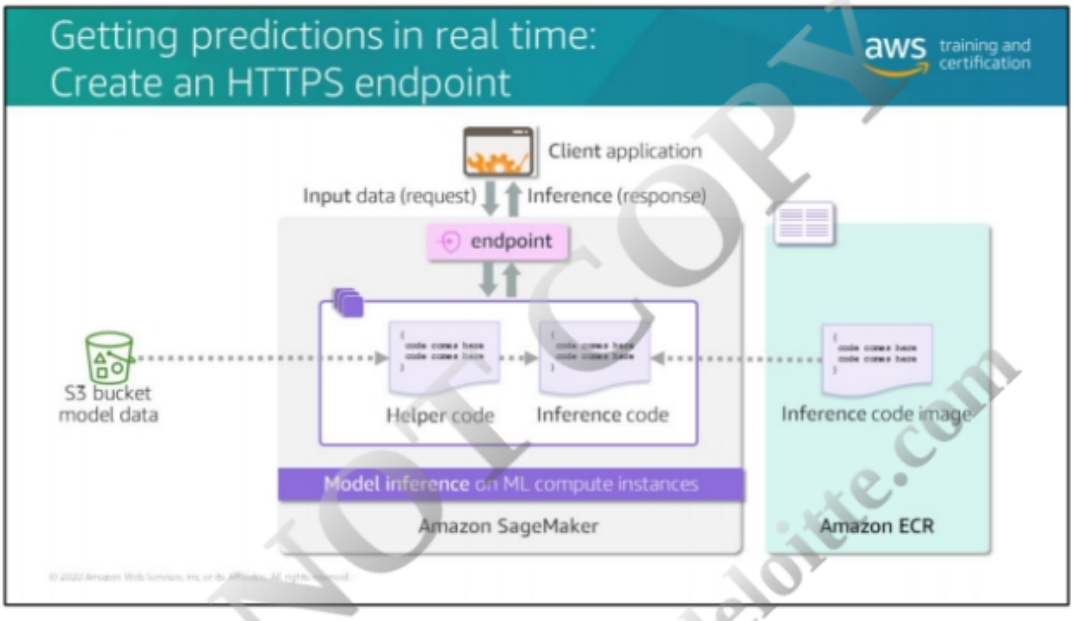
Here's what a batch transform flow would look like (see architecture on slide).



Now that we know how to deploy models and the need to retrain models, let's consider some deployment strategies. We already discussed, hosting a machine learning model in a way that consumers can easily and securely invoke the model to make predictions from it, with low latency. Low latency is particularly important when making real-time predictions, for instance when your customer asks for music recommendations, they expect a response within a matter of seconds.



What if your customers are relying on you for some music to help them distract their crying toddler, but your recommendation service takes 2 minutes to return a song? That would be a very frustrating customer experience, which is why things like recommendation engines are designed to perform inferences within a matter of a few seconds at most.



Once you provide the endpoint configuration to Amazon SageMaker, the service launches the ML compute instances and deploys the model, or models, as specified in the endpoint configuration details, and provides an HTTPS endpoint. Consumers of the model can then use the endpoint to make inferences. SageMaker hosted endpoints provide near real-time, sub-second latency inferences.

The following figure shows the flow of using model hosting services for model deployment (see architecture on slide).

Inferencing vs. training

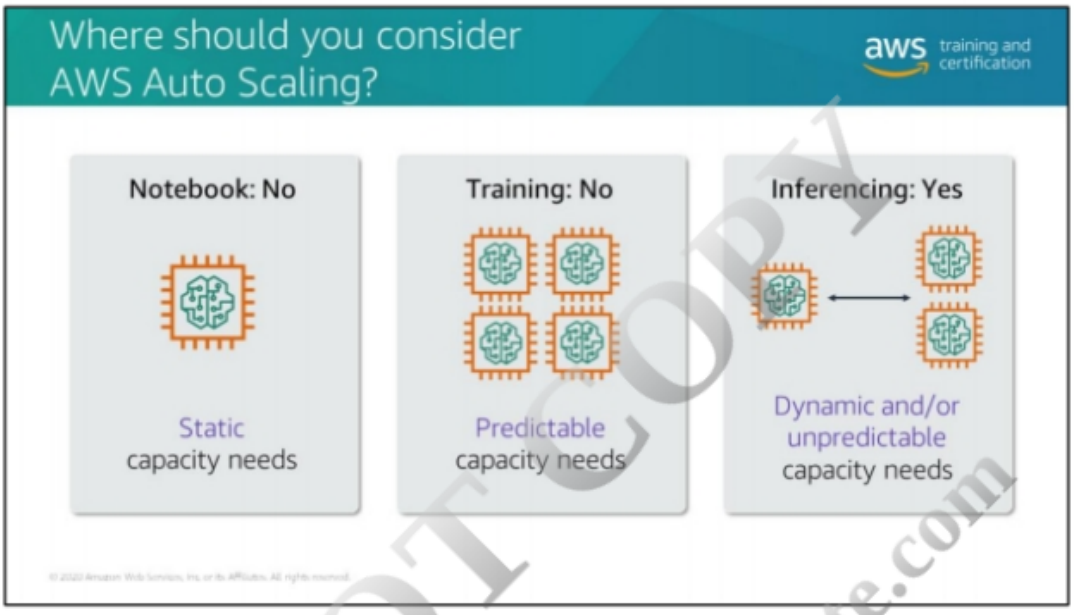
aws training and certification

Inferencing	Training
Usually run on a single input in real time	Requires high parallelism with large batch processing for higher throughput
Less compute-/memory-intensive	Compute/memory intensive
Integrated into the application stack workflows	Standalone, not integrated into an application stack
Runs on different devices at the edge and in the cloud	Run in the cloud
Runs all the time	Typically runs less frequently (train once, repeat infrequently)

Inference is the stage in which a trained model is used to infer or predict the testing samples and comprises of a similar forward pass as training to predict the values. Unlike training, it doesn't include a backward pass to compute the error and update weights.

As a result there are some key differences between inferencing and training.

- Inferencing is usually run on a single input in real-time, whereas training requires high parallelism with large batch processing for higher throughput.
- Inferencing will be less compute and memory intensive than training.
- Inferencing is integrated into the application stack workflows while training is standalone.
- Inferencing runs on different devices at the edge and in the cloud while training runs in the cloud
- Another key difference is that inferencing runs all the time, while training run less frequently because you will likely train infrequently.



When you deploy Amazon SageMaker instances, you should keep in mind that not every instance needs AWS Auto Scaling

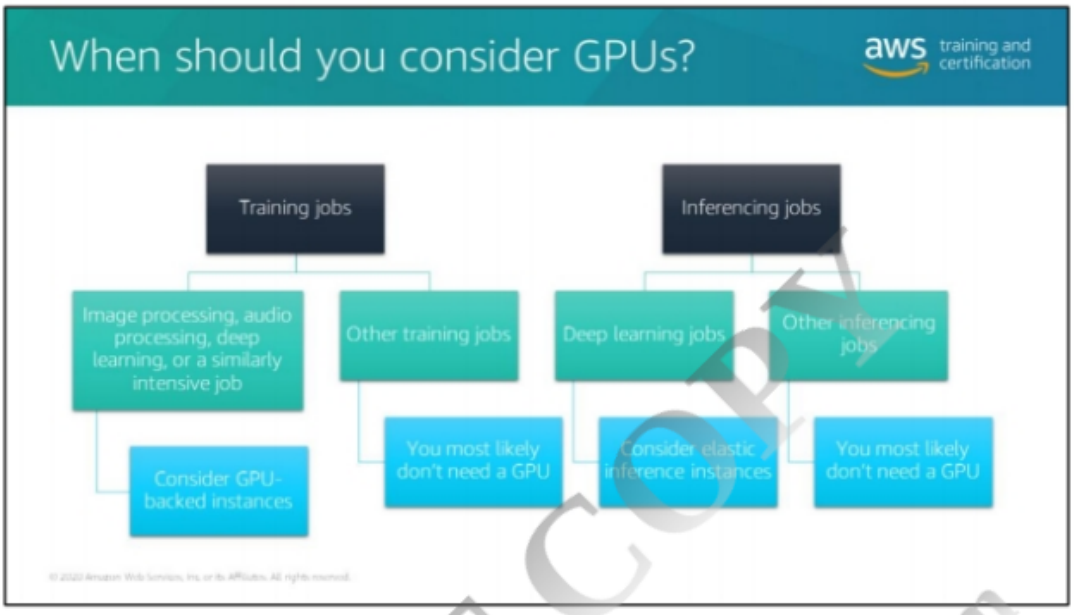
Amazon SageMaker: Instance types

aws training and certification

Instance family	t family	m family	r family	c family	p family	g family	Amazon Elastic Inference
Workload type	Short jobs/ notebooks	Standard CPU to memory ratio	Memory-optimized	Compute-optimized	Accelerated computing-training and inference	Accelerated inference, smaller training jobs	Cost-effective inference accelerators

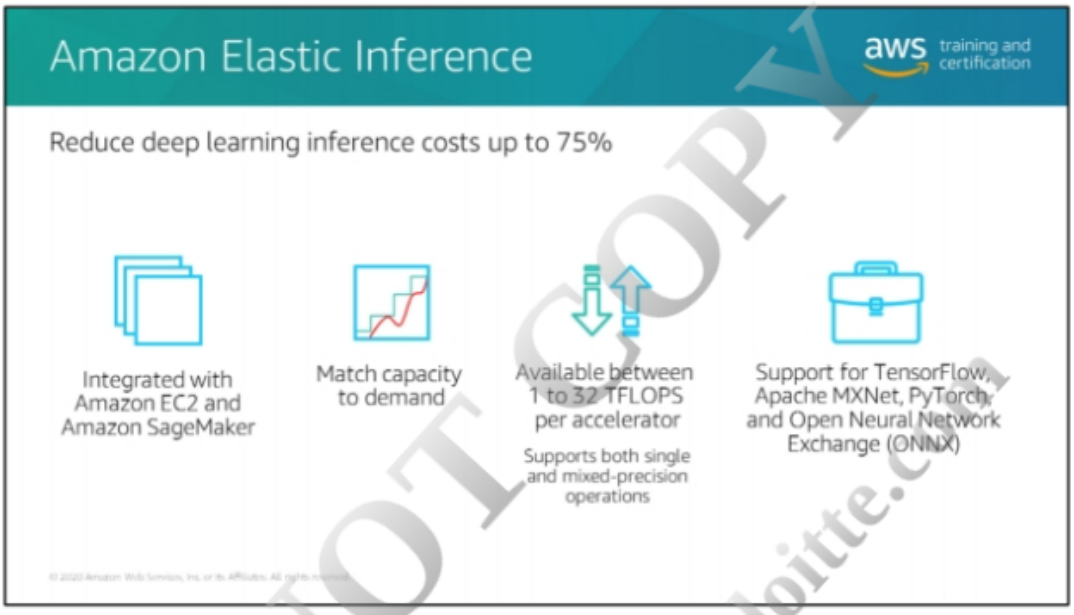
https://aws.amazon.com/sagemaker/pricing/instance-types/

Elastic inference instance types provide GPU-supported compute without the full cost of the more powerful GPU-supported instances.



In addition to the traditional auto scaling of ML compute instances for cost savings, consider the difference between CPU vs GPU. While deep learning based models require high power GPU instance for training, inferences against the deep learning models do not typically need the full power of a GPU. As such, hosting these deep learning models on a full fledged GPU may lead to under utilization and unnecessary costs. **Amazon Elastic Inference** enables you to attach low-cost, GPU-powered acceleration to Amazon EC2 and Amazon SageMaker instances to reduce the cost of running deep learning inferences. Standalone GPU instances are designed for model training and are typically oversized for inference. Even though training jobs batch process hundreds of data samples in parallel, most inference happens on a single input in real-time and consumes only a small amount of GPU compute. Amazon Elastic Inference solves this problem by allowing you to attach the appropriate amount of GPU-powered inference acceleration to any Amazon EC2 or Amazon SageMaker instance type, with no code changes.

(Note : Amazon Elastic Inference is supported in the Elastic Inference-enabled versions of TensorFlow and MXNet. For other deep learning frameworks, use ONNX to export your model, and then import your model into MXNet.)



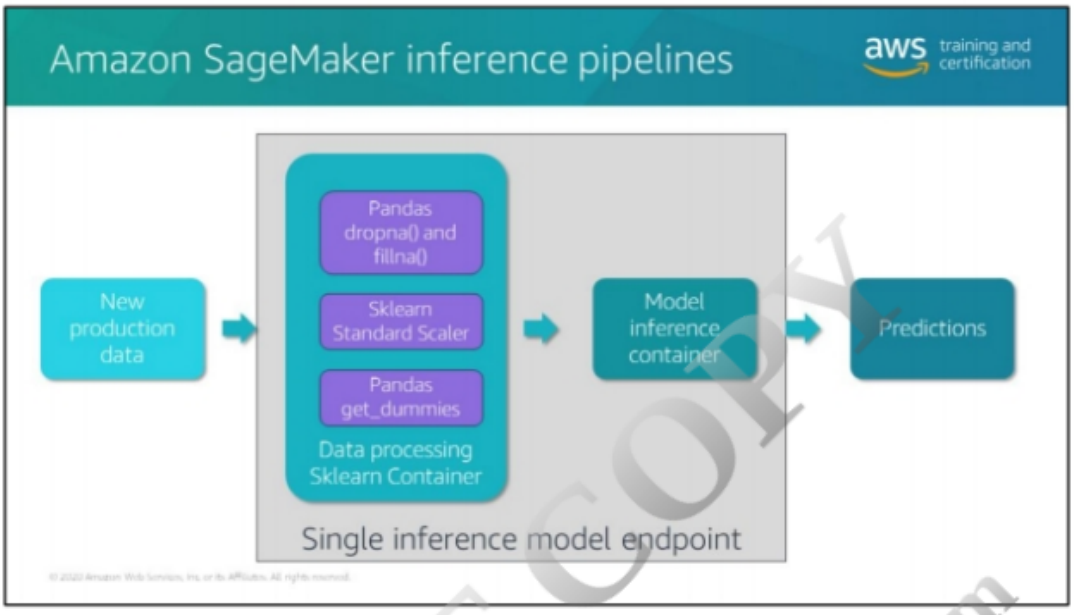
Amazon Elastic Inference allows you to attach low-cost GPU-powered acceleration to Amazon EC2 and Amazon SageMaker instances to reduce the cost of running deep learning inference by up to 75%. Amazon Elastic Inference supports TensorFlow, Apache MXNet, PyTorch, and ONNX models.

In most deep learning applications, making predictions using a trained model—a process called inference—can drive as much as 90% of the compute costs of the application due to two factors. First, standalone GPU instances are designed for model training and are typically oversized for inference. While training jobs batch process hundreds of data samples in parallel, most inference happens on a single input in real-time that consumes only a small amount of GPU compute. Even at peak load, a GPU's compute capacity may not be fully utilized, which is wasteful and costly. Second, different models need different amounts of GPU, CPU, and memory resources. Selecting a GPU instance type that is big enough to satisfy the requirements of the most demanding resource often results in under-utilization of the other resources and high costs.

Amazon Elastic Inference solves these problems by allowing you to attach just the right amount of GPU-powered inference acceleration to any EC2 or SageMaker instance type with no code changes. With Amazon Elastic Inference, you can now choose the instance type that is best suited to the overall CPU and memory needs of your application, and then separately configure the amount of inference acceleration that you need to use resources efficiently and to reduce the cost of running inference.

When you make inferences or predictions against a deployed model, often the raw input data cannot be directly used and it must be preprocessed. As we discussed earlier, this is because machine learning models expect data to be in a predefined format and so the raw data must be first cleaned and preprocessed. For example, if the personalized recommendation algorithm in the eCommerce application can only accept numerical data and if input data is a string or a categorical value, it needs to be converted to numerical format before it can be used.

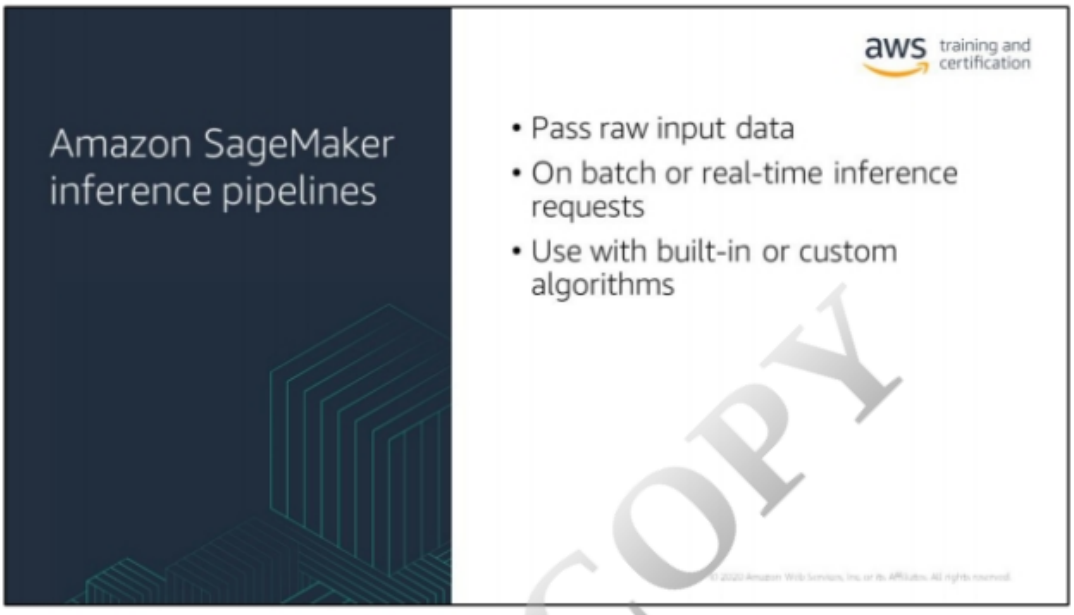
In other cases, combining multiple input features into a single feature can result in more accurate machine learning models. For example, using a combination of temperature and humidity to predict flight delays produces more accurate models.



When you deploy machine learning models into production to make predictions on new data, you need to ensure that the same data processing steps that were used in training are also applied to each inference request. Otherwise, you can get incorrect prediction results.

Using inference pipelines, can reuse the data processing steps applied in model training during inference without the need to maintain two separate copies of the same code. This ensures accuracy of your predictions and reduces development overhead.

Also remember the managed service aspect of Amazon SageMaker. Inference pipelines are completely managed, which means when you deploy the pipeline model, the service installs and runs the sequence of containers on each Amazon EC2 instance in the endpoint or batch transform job. Additionally, the sequence of feature processing and inferences run with low latency because the containers are co-located on the same EC2 instances.



Amazon SageMaker Inference Pipelines enable you to deploy inference pipelines so you can pass *raw* input data, run pre-processing, predictions, and complete post-processing on batch or real-time inference requests. These pipelines are used to define and deploy any combination of pre-trained SageMaker built-in algorithms and your custom ones, packaged in Docker containers. An inference pipeline is a model that is composed of a linear sequence of two to five containers that process requests for inferences on data. You can use these pipelines to combine preprocessing, predictions and post-processing tasks.

